

# PARAMETERIZED OPTICAL VOLUMES

*(WIREFRAMES, MESHES, ETC)*

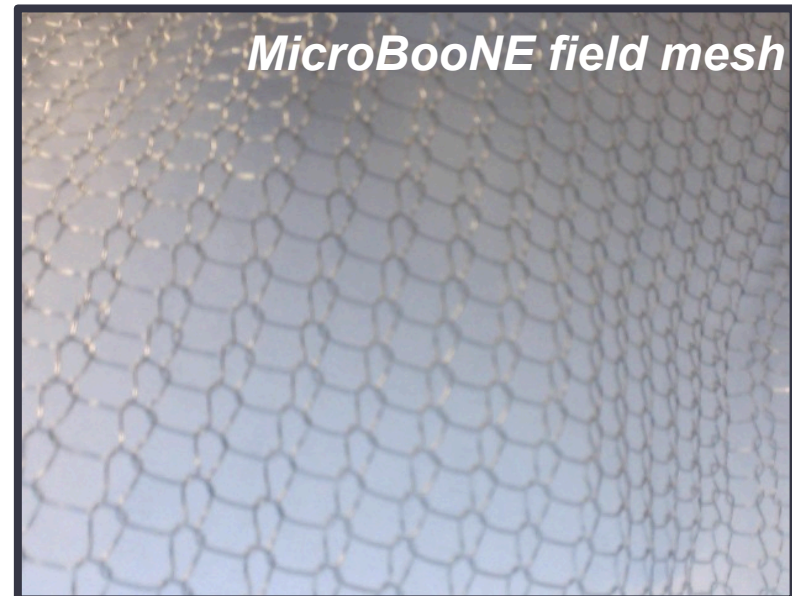
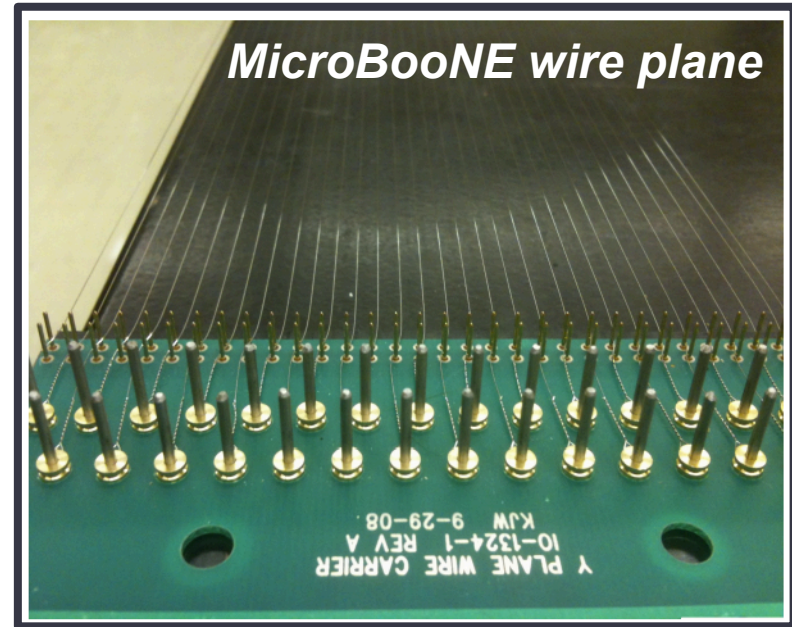
# IN LARSOFT

---

Ben Jones, MIT

# Background

- A while ago, to reduce simulation overhead we decided that the G4 simulation for MicroBooNE would not include a geometrical description of every wire.
- LArG4 also has no “microscopic” definition of the field mesh which sits between wireplanes and PMTs
- For optical simulations, wireplanes and mesh are significant as they absorb photons, and have nontrivial angular transmittances.
- This is also an issue for LBNE paddle detectors, which sit behind wireplane assemblies.
- In this talk I describe an implementation of a very general scheme for simulating partially transparent volumes such as wireplanes in LArG4.



# The new code semi-intuitively:

## root / trunk / LArG4 / OpParamSD.h

History | View | Annotate | Download (2.5 kB)

```
1 ///////////////////////////////////////////////////////////////////
2 /// \file OpParamSD.h
3 //
4 /// \author bjpjones@mit.edu
5 ///////////////////////////////////////////////////////////////////
6 //
7 // This class represents a partially opaque, parameterized optical volume.
8 //
```

*This is a Geant4 object which causes a specified volume to kill photons according to some model*

## root / trunk / LArG4 / OpParamAction.h

History | View | Annotate | Download (3.2 kB)

```
1 // OpParamAction.h - Ben Jones, MIT 2013
2 //
3 // This header file defines various optically parameterized volume actions.
4 //
```

*This file contains several possible photon killing models (and is where you can make new ones)*

## root / trunk / LArG4 / OpDetReadoutGeometry.h

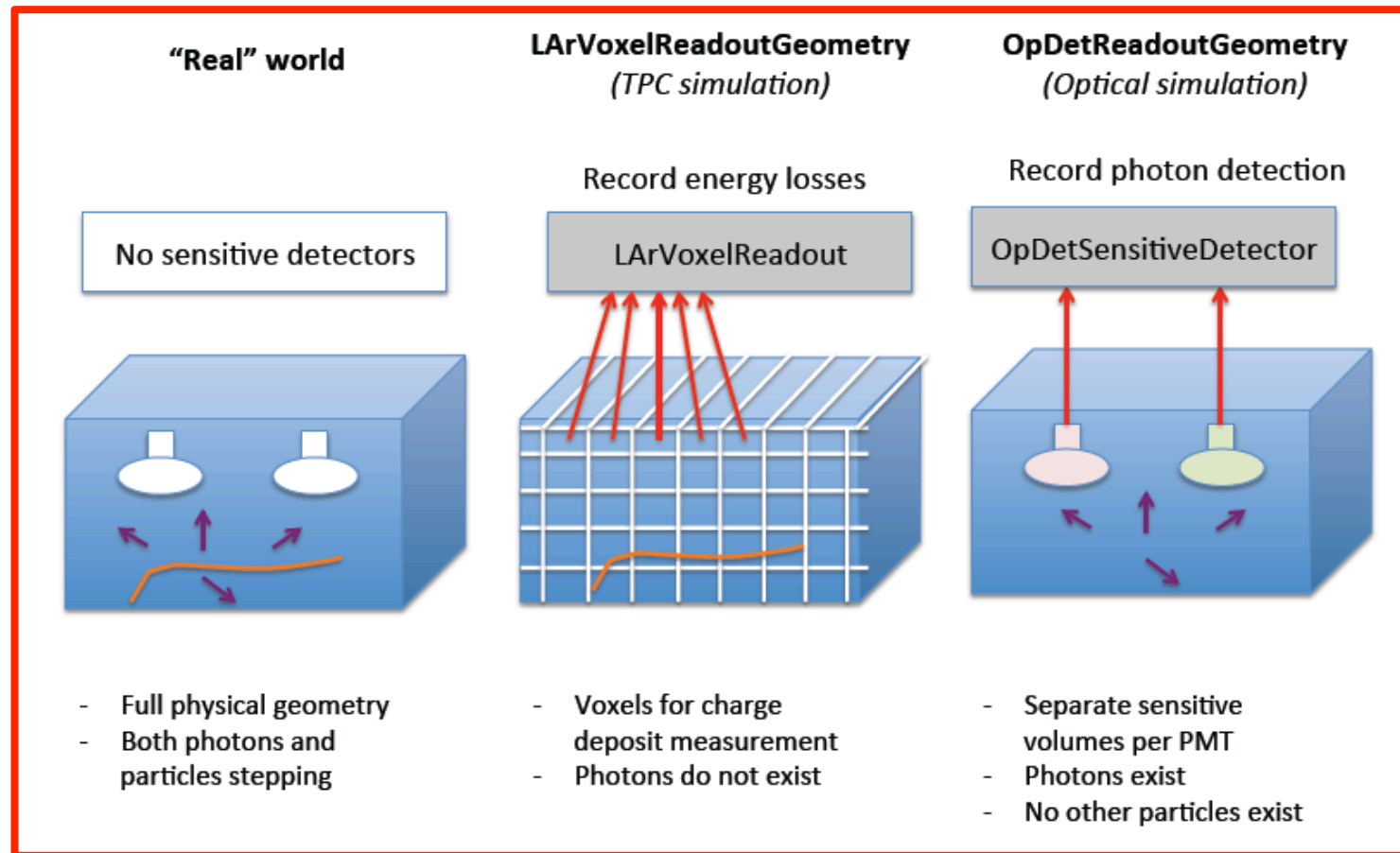
History | View | Annotate | Download (1.8 kB)

```
1 ///////////////////////////////////////////////////////////////////
2 /// \file OpDetReadoutGeometry.h
3 //
4 /// \author bjpjones@mit.edu
5 ///////////////////////////////////////////////////////////////////
```

*This object is the part of LArG4 which interfaces with Geant4 to do most of the optical geometry's "dirty work"*

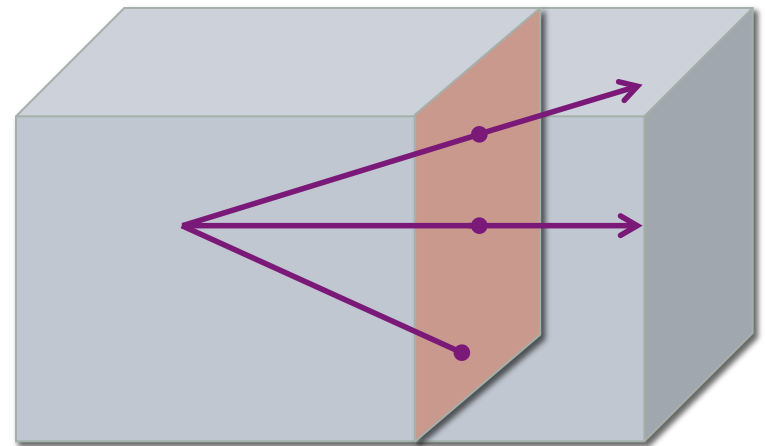
# OpDetReadoutGeometry

- OpDetReadoutGeometry populates a parallel Geant4 world which only optical photons see.



# Adding a New Type of SD

- Now it also places a new type of sensitive detector, whose only job is to selectively kill photons.
- Why make a sensitive detector, rather than a new G4 physics process?
  - 1. We want this to call for every photon in the volume, not just after some mean-free-path
  - 2. We want it to only interact with photons, and not other particles – a sensitive detector can exist in the parallel world which only photons inhabit
- But we want it to be flexible enough to implement different photon killing models for different objects (wires, meshes, non-uniform coatings, etc)



# OpParamSD

- OpParamSD is a G4SensitiveDetector. This is associated to one or more G4PhysicalVolumes at geometry building time.
- When this is done, it is also given a “photon killing model” called fOpa which returns a killing probability as a function of photon position or momentum.
- After this any particle stepping into that volume triggers a method of the OpParamSD, which basically does this:

```
if(G4BooleanRand(fOpa->GetAttenuationFraction(mom,pos)))
{
    // photon survives - let it carry on
    fPhotonAlreadyCrossed[aTrack->GetTrackID()];
}
else
{
    // photon is absorbed
    aStep->GetTrack()->SetTrackStatus(fStopAndKill);
}
```

## OpParamSD Constructor:

Unique name for this Geant4 SD

Name of photon killing model to use

Object orientation (1=x, 2=y, 3=z)

```
OpParamSD::OpParamSD(G4String DetectorUniqueName, std::string ModelName, int Orientation,
std::vector<std::vector<double> > ModelParameters)
: G4VSensitiveDetector(DetectorUniqueName)
{
    // Register self with sensitive detector manager
    G4SDManager::GetSDMpointer()->AddNewDetector(this);

    if(ModelName == "SimpleWireplane")
        fOpa = new SimpleWireplaneAction(ModelParameters, Orientation);

    else if(ModelName == "OverlaidWireplanes")
        fOpa = new OverlaidWireplanesAction(ModelParameters, Orientation);

    else if(ModelName == "TransparentPlaneAction")
        fOpa = new TransparentPlaneAction();

    // else if( your model here )

    else
    {
        mf::LogError("OpParamSD")<<"Error: Optical parameterization model " << ModelName <<" not found"
        assert(0);
    }
}
```

Model parameters (eg wire pitches, diameters)

Choose + setup opacity model

# OpParamAction

Base class:

```
//-----  
// Abstract base class  
//-----  
  
class OpParamAction  
{  
public:  
    OpParamAction();  
    virtual ~OpParamAction();  
    virtual double GetAttenuationFraction(G4ThreeVector PhotonDirection, G4ThreeVector PhotonPosition);  
private:  
};
```

Each derived class  
overrides this  
method

Derived classes:

```
//-----  
// TransparentPlaneAction class  
//-----
```

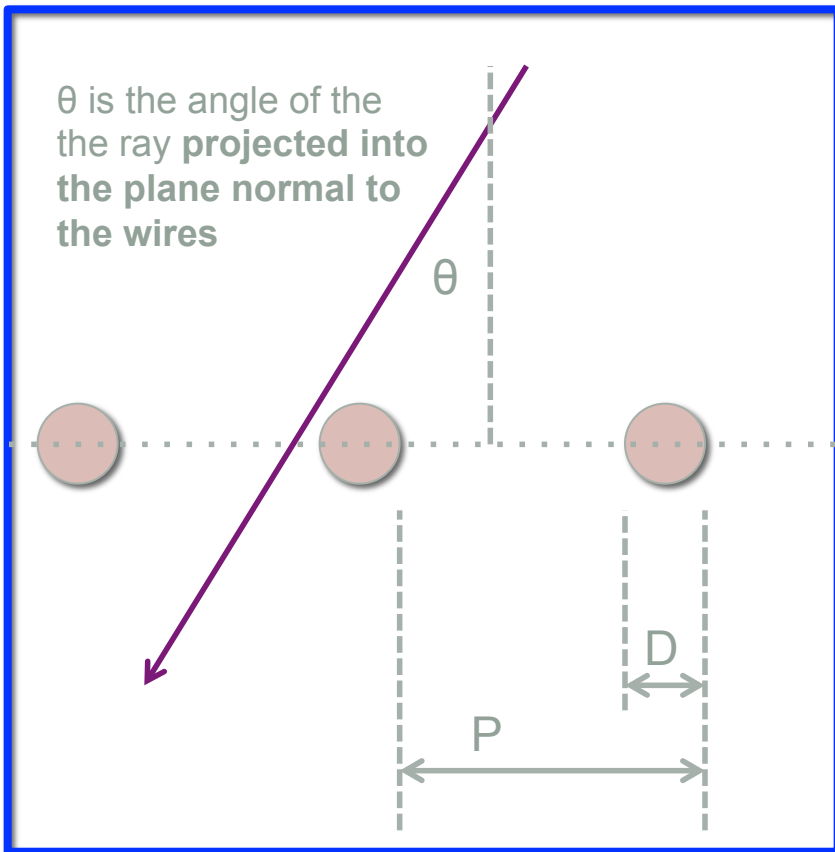
```
//-----  
// SimpleWireplaneAction class  
//-----
```

```
//-----  
// OverlaidWireplanesAction class  
//-----
```

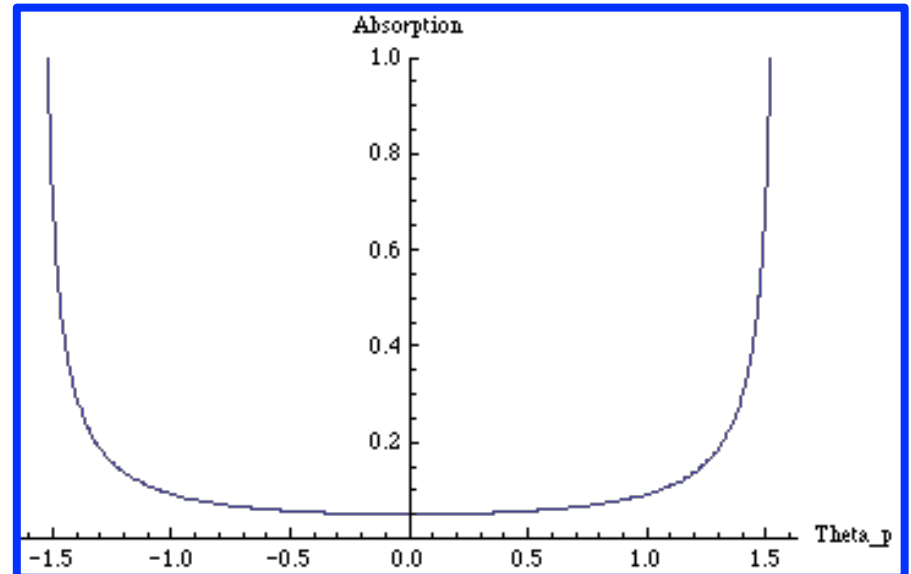
# The Wireplane Action Classes

- Models transmittance as a function of incident light angle on a wireplane of pitch  $P$  and wire diameter  $D$

$\theta$  is the angle of the ray projected into the plane normal to the wires



$$P_{trans} = \begin{cases} 1 - \frac{D}{P \cos \theta} & \cos \theta > \frac{D}{P} \\ 0 & \cos \theta \leq \frac{D}{P} \end{cases}$$



# Configured in LArG4Parameters service

```
# The following parameters specify details of wireplanes or similar
# areas with optically parameterized transmissions (Ben J 2013)

# volume names to be associated with an optical wireplane model
OpticalParamVolumes:      ["volTPCPlaneVert_PV"]      Kill photons in this volume

# specification of which model to use for each volume
OpticalParamModels:       ["OverlaidWireplanes"]      Use overlaid wireplanes model

# orientation of each wireplane set
# 0 = Xdrift, 1 = Ydrift, 2 = Zdrift
OpticalParamOrientations: [0]                        Plane normal is x

# This a set of floats which is specific to the particular model used.
# For overlaid wireplanes, should be a vector of vectors of
#
#                               [plane angle, pitch/mm, wire diameter/mm]
#
# This format is chosen to allow for future extensions to the model
# for, eg, LBNE wireplane development.

#                               List of model parameters:
OpticalParamParameters:   [ [ [30,  3, 0.15],         U
                              [-30, 3, 0.15],         V
                              [0,   3, 0.15],         Z
                              [90,  3.5, 0.2],        Mesh horizontal
                              [0,   3.5, 0.2] ] ]      Mesh vertical
```

**SimpleWireplane**  
implements a single plane

**OverlaidWireplanes**  
Implements several planes superposed (this approximation we use for MicroBooNE)

*Note that all these are vectors, can parameterize any number of different volumes with different (or same) models*

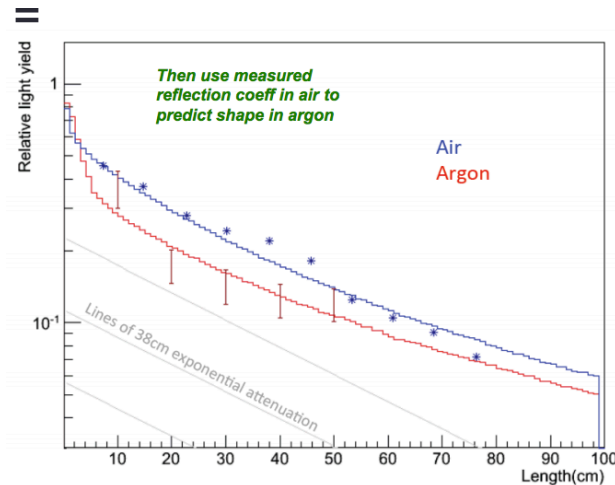
# Extensions to LBNE

- Wireplane absorption should be ~trivially implementable in LBNE using existing model.
- Other applications too: For example, would be easy to write an OpParamAction derived class to model lightguide attenuation
- Atten function supplied either as a list of sensitivity vs length, or some parameterization
- Requires some work, not necessarily best solution, but something to think about.

Attenuation  
volume

Light guide volume

*LGAtten::GetAttenuationFraction  
(TVector3 mom, TVector3 pos)*



# Conclusion

- New simulation tools have been added to LArG4 to model optically parameterized volumes
- Single and multiple wireplanes are ready for action. No need for geometry modifications to use these.
- Production jobs are already being run for MicroBooNE.
- Extension to LBNE wireplanes should be straightforward.
- Possibly wider uses for LBNE (/MicroBooNE?) including paddle attenuation functions
- At some point in next ~2 months I will release a new version of the optical physics technical manual with this feature documented.
- Until then, if you need help please feel free to contact me.